S v e u č i l i š t e u Z a g r e b u
Fakultet strojarstva i brodogradnje

ZAVRŠNI PROJEKT

Jadran Barač

Zagreb, rujan 2007

S v e u č i l i š t e u Z a g r e b u
Fakultet strojarstva i brodogradnje

ZAVRŠNI PROJEKT

UPRAVLJANJE ROBOTOM PRIMJENOM GENETSKIH ALGORITAMA

Voditelj rada:
Dr.sc. Bojan Jerbić

Student:
Jadran Barač

Zagreb, rujan 2007

Sadržaj

Popis slika

Popis tablica

Popis kratica

PSO	Particle swarm optimization
GA	Genetski algoritmi
ACO	Ant colony optimization

1. Uvod

Jedan od glavnih problema u izradi autonomnog robotskog sustava je razvijanje kvalitetnog tehničkog sustava za određivanje putanje. Za autonomnog robota važna je sposobnost planiranja putanje kretanja između neke početne točke i cilja, pritom ne uzrokujući sudare sa preprekama.
Zadatak snalaženja u prostoru koji sadrži prepreke vjerno odražava realnost i čini temeljni problem bilo kojeg autonomnog tehničkog sustava.
Postoji dosta algoritama koji rješavaju ovaj problem no svi imaju bolje i lošije strane. Složenost problema planiranja putanje raste eksponencijalno sa dimenzijama prostorne konfiguracije. Konfiguracija prostora je prostor specifikacija svake točke položaja robotskog sustava. Ovaj se rad temelji na planiranju dvodimenzionalne putanje. Iako je dvodimenzionalni slučaj na mnogo načina najjednostavniji ipak je najviše poučan. Ovakva putanja sa svojom jednostavnošću omogućava znanstvenicima jednostavan uvid kako razni operatori i parametri utječu na robusnost i kvalitetu izlaza algoritma.

2. Pregled metoda za planiranje robotskog ponašanja

2.1 Metaheurističke metode

Riječ heuristika potječe od grčke riječi heurisko što znači pronašao sam. Odavde se da naslutiti da su heuristički algoritmi zapravo algoritmi nastali eksperimentiranjem u svrhu dobivanja zadovoljavajućeg rješenja. Bitno svojstvo heurističkih algoritama je da mogu približno (dovoljno dobro) riješiti probleme eksponencijalne i faktorijelne složenosti. Ipak, valja napomenuti da rješavanje problema heurističkim algoritmima ne mora voditi zadovoljavajućem rješenju, a za neke probleme, heuristički algoritmi pokazuju relativno loše rezultate. To se pogotovo odnosi na probleme za koje postoje egzaktni algoritmi polinomske složenosti. Isto tako, heuristički algoritmi nisu jednoznačno određeni. Pojedini dijelovi heurističkih algoritama se razlikuju ovisno o situaciji u kojoj se koriste. Ti su dijelovi uglavnom funkcije cilja (transformacije) i njihovo definiranje znatno utječe na efikasnost algoritma.

2.1.1 Optimizacija roja čestica

Inteligencija roja (swarm intelligence) je relativno nov pristup u rješavanju problema umjetne inteligencije. Particle swarm optimization (PSO) ili optimizacija roja čestica je stohastička optimizacijska tehnika koju su razvili Dr. Eberhart i Dr. Kennedy 1995. Inspiraciju za razvijanje su dobili na temelju ponašanja i kretanja skupine ptica i riba u jatu. Počelo je kao simulacija pojednostavljenih životinjskih socijalnih sustava, namjera je bila simuliranje kretanja jata riba i ptica. Potom se ustanovilo da se model kretanja životinja u jatu može koristiti kao optimizirajuća tehnika. PSO sadrži mnogo sličnosti sa evolucijskim optimizacijskim tehnikama kao što su genetski algoritmi (GA). Sustav se započinje populacijom nasumičnih rješenja i traži optimum obnavljanjem generacija. Nasuprot GA, PSO ne sadrži evolucijske operatore poput križanja i mutacije. U PSO, moguća rješenja nazivaju se čestice. U usporedbi s GA, prednost PSO je lakša primjena i sadrži manje parametara koji se podešavaju. PSO se uspješno primjenjuje na mnogo područja: učenje umjetnih neuronskih mreža, upravljanje neizrazitih sustava, i ostala područja na koje se GA ne mogu primijeniti.

Pretpostavimo slijedeće okruženje: grupa ptica nasumično traži hranu na nekom području. Samo se jedan komad hrane nalazi u tom području. Ptice ne znaju gdje se hrana nalazi ali svakim pokušajem saznaju koliko su daleko od hrane. Prema tome učinkovita je ona strategija u kojoj se slijedi ptica koja je najbliža hrani. PSO uči iz takvog okruženja i koristi ga za rješavanje optimizacijskih problema. Svako pojedino rješenje je «ptica» u prostoru

pretraživanja. Nazivamo je «čestica». Sve čestice sadrže vrijednosti brzine i dobrote kojima se upravlja letenje čestica nad prostorom rješenja. PSO se inicira grupom nasumičnih čestica (rješenja) i tada traži optimum obnavljanjem generacija. U svakoj iteraciji, sve čestice su obnovljene prema dvjema najboljim vrijednostima. Te vrijednosti su globalno najbolja vrijednost i lokalno najbolja vrijednost. Nakon pronalaženja dvaju najboljih vrijednosti, čestica poboljšava svoju brzinu i položaj. Za vjeran opis ovakvog modela, svakoj čestici je potrebno pridružiti vektor brzine i vektor položaja. Ponekad su potrebne veće dimenzije prostora veće da bi se problem efikasno riješio. Također, svaka čestica pamti svoje najbolje i globalno najbolje rješenje (najbolje rješenje svih čestica). Često se, umjesto globalno najboljeg rješenja, pamti najbolje rješenje susjedne okoline. Za tu svrhu u roju se odrede međusobni susjedi, i svaka čestica pamti najbolje rješenje iz svog susjedstva. Pamćenje najboljeg rješenja iz susjedstva bolje istražuje prostor rješenja, manja je vjerojatnost upadanja u lokalni optimum, ali je konvergencija sporija. Isto tako, lako je uočiti da je globalno najbolje rješenje poseban slučaj najboljeg rješenja iz susjedstva u kojoj su sve čestice međusobno susjedi.

Jedna od prednosti PSO jest uzimanje realnih brojeva za čestice. Pri nalaženju rješenja za jednadžbu *f(x) = x1^2 + x2^2+x3^2* čestica se može postaviti kao (*x1, x2, x3*), a dobrota je *f(x).* Tada se koristi uobičajeni postupak traženja optimuma. Postupak se ponavlja sve dok se ne dostigne maksimalni broj ponavljanja ili dok uvjet minimalne pogreške nije zadovoljen.

Lista podesivih parametara u PSO i njihove normalne vrijednosti:

Broj čestica: uobičajen raspon je 20 do 40. Za većinu problema potreban broj čestica je 10. Za posebno teške probleme uzima se od 100 pa do 200.
Veličine čestica: definira se problemom koji se treba riješiti.
Raspon čestica: također se određuje problemom za optimizaciju. Mogu se navesti različiti rasponi za različite vrijednosti čestica.
Vmax: određuje maksimalnu promjenu koju čestica može izvršiti tijekom jednog ponavljanja. Obično uzimamo raspon čestice kao *Vmax*, na primjer: čestici (*x1, x2, x3*) *x1* pripada raspon [-10, 10], tada je *Vmax* 20.
Faktori učenja: za faktore učenja obično se uzima 2 i ima raspon [0, 4] no to opet ovisi o problemu.
Uvjet zaustavljanja: ovisi o broju ponavljanja (iteracija) i o dozvoljenoj veličini pogreške.

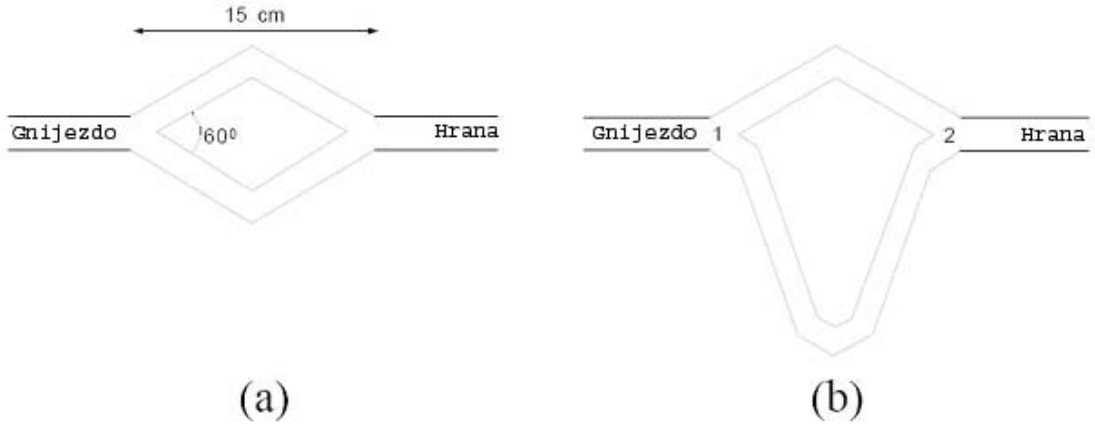
Postoje dvije verzije PSO, globalna i lokalna verzija. Globalna verzija je brža ali za neke probleme lakše upada u lokalni optimum. Lokalna verzija je nešto sporija ali se lako ne zaustavlja u lokalnom optimumu.

2.1.2 Optimizacijski algoritam mravlje kolonije

Ponašanje mrava u kolonijama su izvor brojnih metoda i tehnika. Ant colony optimization (ACO) ili optimizacijski algoritam mravlje kolonije je

inspiriran ponašanjem mrava u potrazi za hranom. Mravi ostavljaju feromone na tlo kako bi označili najpovoljnije staze u potrazi za hranom. Ostali članovi kolonije opažaju prisutnost feromona i nastoje pratiti one staze na kojima je koncentracija feromona veća. Optimizacijski algoritam mravlje kolonije rabi slične mehanizme za rješavanje optimizacijskih problema.

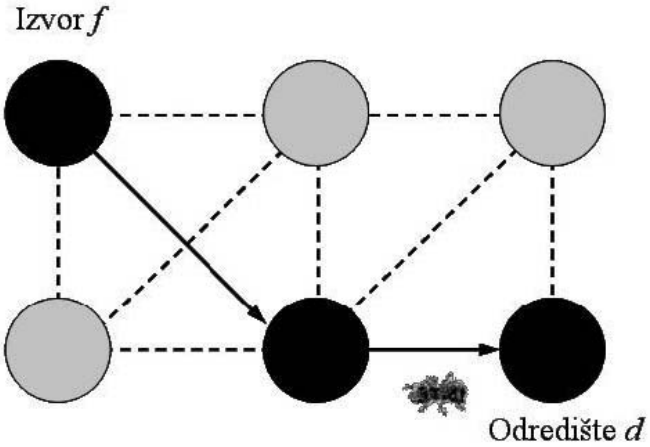
U eksperimentu zvanom “dupli most”provedeno je istraživanje ponašanja mrava u skladu sa postavljenim feromonima. Mravlje gnijezdo je povezano prema izvoru hrane sa dva mosta jednake duljine. U takvoj situaciji mravi odmah istražuju okolinu gnijezda i na kraju pronađu hranu. Na putu između hrane i gnijezda mravi nanose feromone. U početku svaki mrav nasumice izabire jedan od dva mosta. Međutim nakon određenog vremena jedan od dva mosta predstavlja veću koncentraciju feromona i prema tome privlači veći broj mrava. To vodi da cijela kolonija na kraju konvergira prema upotrebi jednog mosta. Korištenjem pozitivne povratne informacije mravi mogu rabiti najkraći put između hrane i gnijezda. U drugoj varijanti eksperimenta “dupli most” u kojoj je jedan most znatno duži od drugog, kako je prikazano prema slici 1.



Slika 1. Eksperimentalna struktura za dupli most. (a) Grane imaju jednaku duljinu. (b) Grane imaju različitu duljinu

U ovoj varijanti prvobitno kolebanje u izboru mosta se značajno smanjuje. Mehanizam koncentracije feromona ima veliku ulogu: mravi koji slučajno izaberu kraći most su prvi koji će se vratiti u gnijezdo. Kraći most, prema tome, ranije dobiva feromone i time povećava vjerojatnost da ga ostali mravi izaberu prije nego ovaj drugi.

Analogno prirodnim mravima, zadatak je umjetnih mrava pronaći minimalni put između čvorova grafa. Neka je $G = (N, A)$ povezani graf, gdje je N skup svih čvorova, a A skup svih lukova. Broj čvorova je $N = n$. Rješenje problema predstavlja put na grafu koji povezuje izvorišni čvor f s odredišnim d, čija je duljina određena brojem lukova na putu. Slika 2. prikazuje minimalni put na grafu.



Slika 2 Pronalazak minimalnog puta na grafu.

Sa svakim je lukom (i, j) grafa *G* povezana varijabla τ_{ij} koja predstavlja umjetni trag feromona. Tragove feromona čitaju i pišu mravi. U svakom čvoru grafa dolazi do stohastičke odluke koji je čvor sljedeći. *k* -ti mrav lociran u čvoru *i* koristi trag feromona τ_{ij} za izračun vjerojatnosti u koji čvor $j \in N_i$ treba otići. N_i je skup susjeda čvora *i* . Vjerojatnost odabira čvora *j* je dana izrazom (2.1).

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}}{\sum_{j \in N_i} \tau_{ij}} & , j \in N_i \\ 0 & , j \notin N_i \end{cases} \qquad (2.1)$$

Dok traži rješenje mrav ostavlja trag feromona na pripadnom luku. Međutim, tako definirano rješenje može dovesti do konvergencije prema suboptimalnom rješenju tako da se uvodi mehanizam isparavanja. U svakoj se iteraciji eksponencijalno smanjuje količina feromona. Ponašanje algoritma ovisi o broju susjeda svakog čvora. Ukoliko čvorovi grafa imaju više od dva susjeda, algoritam gubi na stabilnosti. Odnosno, postaje kritičan izbor parametara. Algoritam je moguće poboljšati sagledavanjem svojstava kolonije s jedne strane te sagledavanjem svojstava pojedinog mrava.

2.1.2.1 Svojstva kolonije

Algoritam je moguće poboljšati korištenjem heuristike. Moguće je heuristički mijenjati količinu ostavljenog feromona tako da je ona proporcionalna kvaliteti rješenja koje se generira. Za pronalazak najkraćeg puta koji obilazi svaki čvor samo jednom, mravi moraju imati barem ograničenu memoriju.

Svojstva kolonije kao cjeline:

1) Dobra rješenja mogu proizaći samo kao rezultat kolektivne interakcije;
2) Svaki mrav koristi samo privatne informacije i lokalne informacije čvora kojeg posjećuje;
3) Mravi komuniciraju samo indirektno;
4) Mravi se sami po sebi ne adaptiraju, naprotiv, mijenjaju način prezentacije problema i percepcije drugih mrava.

2.1.2.2 Svojstva mrava

Svojstva su mrava kao individue:

1) Mrav traži najisplativije rješenje;
2) Svaki mrav ima memoriju M_k za pohranu informacija o putu i može ju koristiti:
    a. za generiranje smislenog rješenja,
    b. za evaluaciju pronađenog rješenja
    c. za povratak po istom putu ;
3) Mrav *k* može se pomaknuti u bilo koji čvor *j* u susjedstvu N_i^k
4) Mravu *k* može se dodijeliti početno stanje jednog ili više terminirajućih uvjeta;
5) Mrav stvara rješenje postupno i konstrukcija završava kada je zadovoljen jedan od uvjeta zaustavljanja;
6) Odabir sljedećeg čvora temelji se na vjerojatnosti;
7) Vjerojatnost pojedine odluke mrava temelji se na:
    a. vrijednostima pohranjenim u strukturi čvora $A_{ij} = [\mathrm{a_{ij}}]$ (tablica ruta) koja se sastoji od kombinacije lokalnih tragova feromona i vrijednosti heurističke funkcije,
    b. privatnoj memoriji mrava,
    c. ograničenjima problema;
8) Pomicanjem iz čvora *i* u čvor *j* mrav ažurira trag feromona;
9) Kada je izgradio rješenje mrav se može vratiti istim putem i ažurirati trag feromona na povratku;
10) Kada je izgradio rješenje i vratio se na izvorište, mrav umire i oslobađa resurse.

2.1.2.3 Primjene optimizacije kolonijom mrava

Primjene optimizacije kolonijom mrava obuhvaćaju NP-teške probleme. NP-teški problemi su oni optimizacijski problemi kod kojih se složenost ne može izraziti kao polinomska ovisnost. Npr. kod pronalaska minimalnog puta u grafu. Graf je eksponencijalno ovisan o dimenziji problema. U ovom slučaju mravi koriste puno manji graf koji je sastavljen od segmenata problema. Nakon mrava za dobivanje rješenje se može koristiti optimizator za specifičan problem kako bi pronašao lokalni minimum. Optimizaciju kolonijom mrava je moguće koristiti i kod problema najkraćeg puta gdje se graf dinamički mijenja za vrijeme optimizacije. Kolonija je mrava prigodna i za rješavanje različitih problema koji nastaju kod upotrebe distribuirane računalne arhitekture.

3. Ostale metode

3.1 Metoda potencijalnog polja

Metoda potencijalnog polja se primjenjuje za upravljanje robotom. Uobičajena metoda potencijalnog polja za izbjegavanje prepreka sadrži procjenu odbojne sile (3.2) (repulsive force) i privlačne sile (3.1) (attractive force) kojom robot teži prema svom cilju. Prepreka je promatrana kao točka najvišeg potencijala, konačna željena pozicija je promatrana kao točka najnižeg potencijala. Takva procjena je napravljena uzimajući u obzir udaljenost i relativnu brzinu između robota i prepreka. Svaka od ovih sila sadrži smjer objekta od kojeg potječe. Privlačna sila ubrzava robot prema cilju dok odbojna sila ubrzava robot u suprotnom smjeru od cilja.

$$F_a = \frac{1}{D_{rg}} \qquad (3.1)$$

$$F_r = \frac{1}{(pD_{ro})^n} \qquad (3.2)$$

Privlačnu silu predstavlja F_a, odbojnu silu F_r. D_{rg} je udaljenost između robota i cilja, D_{ro} je udaljenost između robota i prepreke; *p* i *n* predstavljanju parametre za podešavanje. Zamisao koja se krije iza metode potencijalnog polja je analogija s kretanjem električno nabijenih čestica u prostoru: svaka se odbija od čestica sa jednakim nabojem a privlači sa česticama suprotnog naboja. Sile koje nastaju međudjelovanjem naelektriziranih čestica sadrže pravac tih čestica. Smjer pravca je suprotan od čestice ako dotična čestica u međudjelovanju ima isti naboj, ako ima suprotni naboj tada obrnuto. Jačina elektrostatičke sile ne ovisi o brzini čestica, budući je polje radijalno, dovoljno je znati njihovu međusobnu udaljenost kako bi se potpuno definirala funkcija potencijala. Kretanje se izvodi ponavljanjem u kojem se pobuđuje umjetna sila prema(3.3).

$$\vec{F}(q) = -\vec{\nabla}U(q) \qquad (3.3)$$

Ova sila usmjerava robot da se kreće u smjeru opadanja potencijalnog polja. Gdje je $\vec{\nabla}$ gradient u odnosu prema *q*. *q=(x,y)* predstavlja koordinate položaja robota. Korištenjem najjednostavnijeg oblika metode potencijalnog polja pri navigaciji, robot će često biti u škripcu tj. zaglaviti će se. To je zbog prisutnosti lokalnog optimuma u potencijalnom polju iz kojeg robot ne može izaći. Taj se problem rješava brojnim tehnikama optimizacije koje omogućavaju daljnji razvoj i usavršavanje ove metode kontrole i navigacije robota.

3.1.1 Problemi kod metode potencijalnog polja

U eksperimentalnom radu sa metodom potencijalnog polja identificirala su se 4 značajna problema :

1. Često zarobljavanje u lokalnom optimumu (ciklično ponašanje)
   Najčešći i najpoznatiji problem kod metode potencijalnog polja su problemi lokalnog optimuma ili tzv. situacije zaglavljenja. Problemi nastaju kad robot uđe u slijepu ulicu (prepreka U-oblika). Takva zaglavljenja mogu nastati na više različitih oblika prepreka. Iako zaglavljenja se mogu riješiti heurističkim ili globalnim povratom (global recovery).
   Kod heurističkih povrata rješenja nisu optimalna, zbog toga pristup heurističkih povrata je napušten.

2. Ne nalazi prolaz kroz usko razmaknute prepreke
   Odbojne sile dviju usko razmaknutih prepreka su izmiješane u dvije nezgrapne odbojne sile. Suma tih dviju nezgrapnih sila je usmjerena nasuprot otvora pa se robot okreće i udaljava od prolaza između tih prepreka.

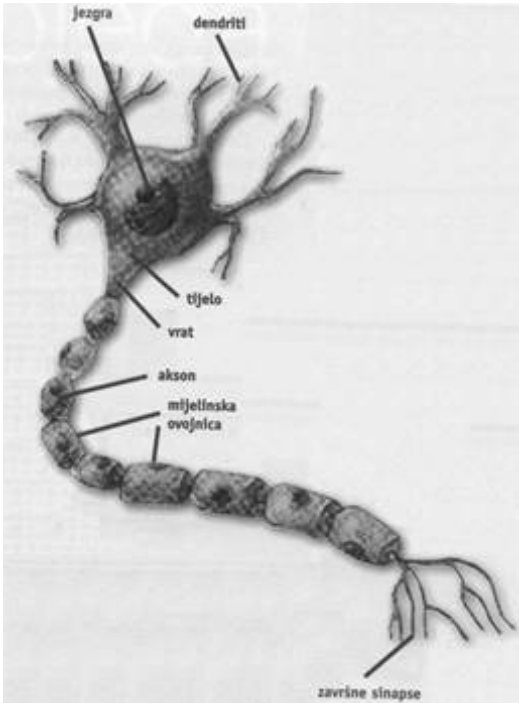
3. Oscilacije u prisustvu prepreka
   Jedno od najvećih ograničenja metode potencijalnog polja je težnja ka nestabilnim kretnjama u prisustvu prepreka.

4. Oscilacije u uskim prolazima
   Sličan iako ozbiljniji problem nastaje kada robot prolazi kroz uske prolaze, u kojima je robot simultano izložen odbojnim silama sa suprotnih strana pa je njegova putanja narušena.

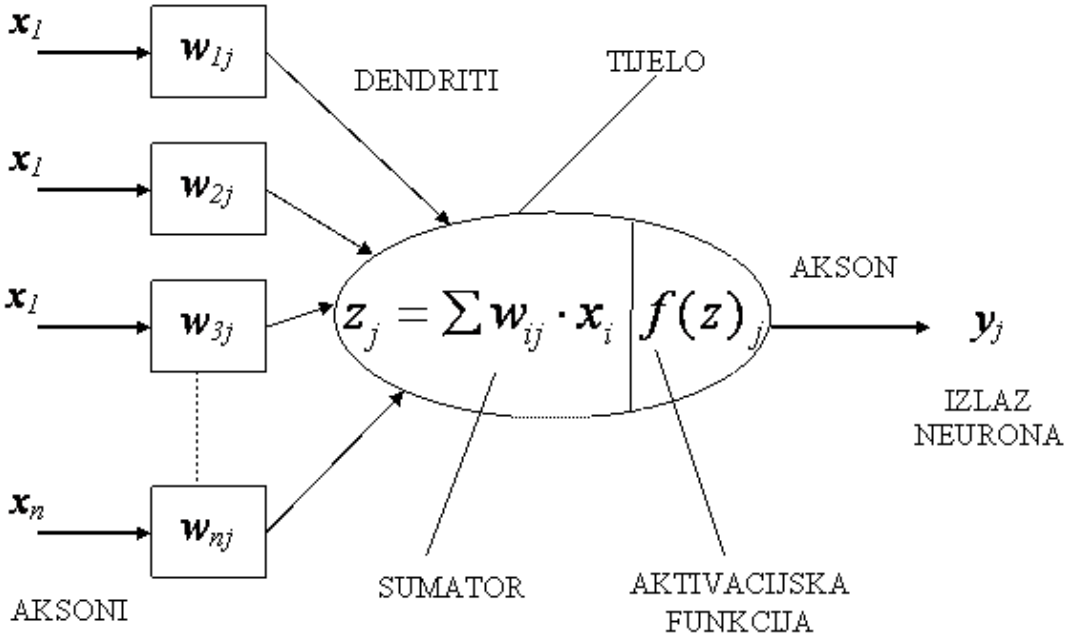
Za brzo pronalaženje rješenja u izbjegavanju prepreka zbog navedenih problema metoda potencijalnog polja je napuštena.

3.2 Neuronske mreže

Za umjetni neuron može se reći da se dizajnira s idejom da oponaša osnovne funkcije (karakteristike prvog reda) biološkog neurona. Tijelo biološkog neurona zamjenjuje se sumatorom, ulogu dendrita preuzimaju ulazi u sumator, izlaz sumatora je akson umjetnog neurona, a uloga praga osjetljivosti bioloških neurona preslikava se na tzv. aktivacijske funkcije.



Slika 3. Struktura biološkog neurona



Slika 4. Struktura umjetnog neurona

Funkcijske sinaptičke veze biološkog neurona s njegovom okolinom preslikavaju se na težinske faktore, preko kojih se i ostvaruje veza umjetnog neurona s njegovom okolinom. Težinski faktor može biti pozitivan ili negativan broj, a kod suvremenih umjetnih neuronskih mreža i neka funkcija (varijabilan težinski faktor). Kad je težinski faktor jednak nuli onda odgovarajuća veza s okolinom neurona ne postoji, pa se u shemi neuronske mreže i ne ucrtava. Težinski faktori rade isto ono što i sinapse kod biološkog neurona: povezuju izlaze iz okoline neurona, odnosno izlaze drugih neurona (aksone) s ulazima sumatora (dendriti). Intenzitet te veze ovisi o iznosu (modulu), a karakter veze o predznaku težinskog faktora. Izlaz sumatora povezuje se na ulaz aktivacijske funkcije, koja na svom izlazu producira izlaz umjetnog neurona. Aktivacijske funkcije mogu biti linearne i nelinearne*.* Kod linearnih, izlaz sumatora množi se s nekim faktorom (pojačanjem) i tako dobiva izlaz neurona. Nelinearne aktivacijske funkcije mogu poprimiti različite oblike, ali se najčešće koriste*:* funkcije praga osjetljivosti, sigmoidalne, hiperbolične i harmoničke funkcije**.** Nelinearne aktivacijske funkcije prevode izlaz sumatora na izlaz neurona preko nelinearnog pojačanja*.* Tako funkcija praga osjetljivosti daje na izlazu neurona jedinicu, ako je izlaz sumatora veći od nekog zadanog broja (prag osjetljivosti), što odgovara ispaljivanju impulsa kod biološkog neurona. U suprotnom slučaju izlaz neurona je nula (neuron nije aktivan).

Prema izloženom, može se zaključiti da umjetni neuron funkcionira slično kao biološki neuron. Izlazi iz drugih neurona i/ili okruženja promatranog neurona, koji se upućuju promatranom neuronu, množe se težinskim faktorima i dovode do sumatora . U sumatoru se tako dobiveni produkti sumiraju a njihova suma se odvodi na ulaz aktivacijske funkcije, koja će na svom izlazu dati izlaz neurona. Tako na primjer, ako je izlaz sumatora

označen sa *x*, izlaz neurona sa *y*, a aktivacijska funkcija je oblika sinusa, vrijedit će izraz *y=sin(x).*

Ovisno o postavljenom kriteriju mogu se dobiti različite kategorizacije umjetnih neuronskih mreža. Paralelno složen skup neurona gradi jedan sloj neuronske mreže. Umjetne neuronske mreže mogu biti jednoslojne i višeslojne. Uobičajeno je da višeslojne neuronske mreže imaju ulazni i izlazni sloj, a između njih su tzv. skriveni slojevi. Ako se slojevi neuronske mreže povežu tako da signali putuju samo u jednom smjeru, od ulaza ka izlazima, onda govorimo o **unaprijednim** neuronski mrežama (feedforward neural networks). Ukoliko postoji bar jedna povratna petlja (suprotni smjer signala) radi se o tzv. **povratnim** neuronskim mrežama(recurrent neural networks)

3.2.1 Kako uče umjetne neuronske mreže

U principu se razlikuje **supervizorno** (uz nadzor) i **nesupervizorno** (bez nadzora) učenje neuronskih mreža. Supervizorno učenje zahtijeva vanjskog «učitelja» neuronske mreže, koji promatra ponašanje mreže korigirajući istu dok se ne dobije željeno ponašanje mreže. Kod ovog načina najprije se usvoji određena struktura mreže (broj ulaza, broj neurona, broj slojeva, broj izlaza, te broj težina mreže). Usvoje se, obično preko generatora slučajnih brojeva, početne težine neuronske mreže. Zatim se na ulaz mreže dovodi skup ulaznih varijabli. Mreža producira odgovarajući skup izlaznih varijabli. Skup izlaznih varijabli uspoređuje se sa željenim skupom izlaznih varijabli. Razlika željenih i stvarnih izlaza neuronske mreže gradi pogrešku mreže, koja se koristi za računanje novih težina (parametara) preko određenog (usvojenog) algoritma. Cijeli postupak ponavlja se iteracijski dok pogreška mreže ne bude manja od unaprijed zadanog iznosa. Pritom se, prema potrebi mijenja struktura mreže(broj neurona, broj slojeva, broj težina). Nakon procesa učenja (treninga) slijedi proces testiranja neuronske mreže. To se radi s novim skupom ulaza mreže koji nije bio sadržan u ulaznom skupu za vrijeme procesa učenja. Mreža sada producira nove izlaze koji se uspoređuju sa željenim izlazima. Pritom se ne mijenjanju parametri (težine) mreže. Iznos pogreške mreže u procesu testiranja služi za ocjenu **robusnosti**, odnosno generalizacijskih svojstava mreže, tj. sposobnosti mreže da daje zadovoljavajuće izlaze (rezultate) i za skup ulaza kojim nije bila učena. Kod nesupervizornog učenja neuronske mreže ne koristi se vanjski učitelj. Ovdje se neuronska mreža sama organizira, pa se mreže učene ovom metodom nazivaju samoorganizirajuće neuronske mreže. Na ulaz mreže dovodi se skup ulaznih varijabli, mreža se samoorganizira podešavanjem svojih parametara (težina) po dobro-definiranom algoritmu. Budući da željeni izlaz mreže nije specificiran za vrijeme za vrijeme učenja mreže, rezultat učenja nije predvidiv. Nakon učenja provodi se postupak testiranja.

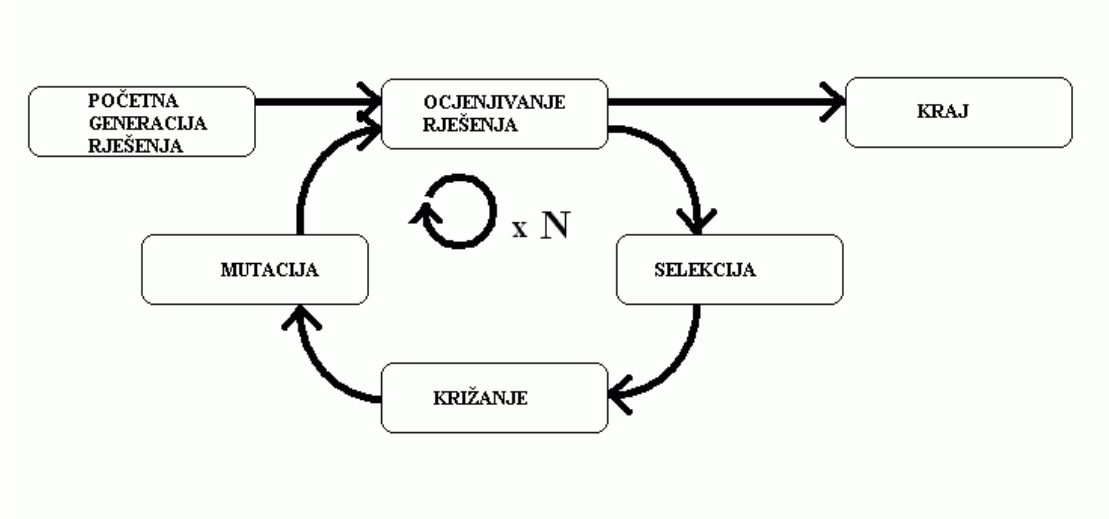
3.2.2 Primjena neuronskih mreža u robotici

Umjetne mreže imaju izuzetno široko područje primjene. Jedno od najvažnijih područja primjene umjetnih neuronskih mreža jest upravo robotika. Naime, na suvremene robote postavljaju se visoki zahtjevi u smislu povećanja brzine rada i točnosti (paralelno procesiranje signala), snalaženja u neorganiziranoj okolini (robotski vid), prirodne komunikacije sa okolinom (robotski sluh, vid i govor) te donošenje odluka na osnovi percepcije svog stanja i stanja okoline (umjetna inteligencija). S dinamičke točke gledišta, robot je izuzetno složen nelinearni dinamički sustav. Vezano s tim i upravljački algoritmi imaju složene strukture upravljanja. Primjena konvencionalne kompjutorske tehnike, općenito gledano, ne može zadovoljiti zahtjeve koji se postavljaju na brzinu izvođenja upravljačkih algoritama robota. Radi povećanja brzine reagiranja robota na promjene stanja robota i njegove okoline, te praćenja željene trajektorije, zahtijeva se paralelna struktura algoritma upravljanja (vođenja), koja omogućava istovremeno izračunavanje upravljačkih varijabli za sve stupnjeve slobode gibanja robota. Takva struktura upravljanja može se relativno jednostavno realizirati primjenom umjetnih neuronskih mreža kao sustav za vođenje robota, budući da je jedna od glavnih karakteristika tih mreža upravo paralelno procesiranje signala s ulaza na izlaz mreže. Danas su već razvijene neuronske mreže za **identifikaciju dinamike robota**, rješavanje tzv. inverznog kinematičkog problema, automatsko **planiranje trajektorija** robota, optimalno i adaptivno vođenje robota u prostoru i vremenu.

Jednostavno je zaključiti da je na području istraživanja, usavršavanja i primjene umjetnih neuronskih mreža napredak nezaustavljiv, iako se problematika istih i konkretna primjena premalo predstavljaju javnosti, iako se sve više primjenjuju u svakodnevnom životu, za razliku od sve prisutnije računalne tehnologije.

4. Općenito o teoriji genetskih algoritama

Kako nam evolucija može pomoći pri potrazi za rješenjem pretraživanja? Odgovor na ovo pitanje predložen je u sedamdesetim godinama u obliku genetskih algoritama. Genetski algoritmi su grupa algoritama koji koriste evolucijske procese kako bi pronašli najbolje rješenje. Osnovna ideja je da se ograničen broj rješenja bori za opstanak i rekombinaciju uz stalno stvaranje novih rješenja. Ova ideja provodi se po uzoru na evoluciju u prirodi pa se rješenja mijenjaju i unaprjeđuju kroz generacije. U posljednjih petnaestak godina zabilježen je značajan razvoj genetskih algoritama. Genetski algoritam se primjenjuje i daje dobre rezultate u području učenja kod neuronskih mreža, pri traženju najkraćeg puta, problemu trgovačkog putnika, strategiji igara, problemima sličnim transportnom problemu, problemu raspoređivanja procesa, problemu određivanja parametara sustava, optimiranju upita nad bazom podataka, itd.



Slika 5. Shema genetskog algoritma

Genetski algoritam je heuristička metoda optimiranja koja imitira prirodni evolucijski proces. Evolucija je robustan proces pretraživanja prostora rješenja. Živa bića se tijekom evolucije prilagođavaju uvjetima u prirodi, tj. životnoj okolini. Analogija evolucije kao prirodnog procesa i genetskog algoritma kao metode optimiranja, očituje se procesu selekcije i genetskim operatorima. Mehanizam odabira nad nekom vrstom živih bića u evolucijskom procesu čine okolina i uvjeti u prirodi. U genetskim algoritmima ključ selekcije je funkcija cilja, koja na odgovarajući način predstavlja problem koji se rješava. Slično kao što su okolina i uvjeti u prirodi ključ selekcije nad nekom vrstom živih bića, tako je i funkcija cilja ključ selekcije nad populacijom rješenja u genetskom algoritmu. Naime, u prirodi jedinka koja je najbolje prilagođena uvjetima i okolini u kojoj živi ima najveću vjerojatnost preživljavanja i parenja, a time i prenošenja svojega genetskog materijala na

svoje potomke. Za genetski algoritam jedno rješenje je jedna jedinka. Selekcijom se odabiru dobre jedinke koje se prenose u slijedeću populaciju, a manipulacijom genetskog materijala stvaraju se nove jedinke. Takav ciklus selekcije, reprodukcije i manipulacije genetskim materijalom jedinki ponavlja se sve dok nije zadovoljen uvjet zaustavljanja evolucijskog procesa. Genetski algoritmi predloženi su od strane Johna H. Hollanda još u ranim sedamdesetima. Tijekom posljednjih nekoliko godina, pokazali su se vrlo moćnim i u isto vrijeme općenitim alatom za rješavanje čitavog niza problema iz inženjerske prakse. To se može objasniti njihovom jednostavnošću; kako same ideje na kojoj su osnovani, tako i njihove primjene; te doprinosu niza znanstvenika i inženjera na njihovom prilagođavanju velikom broju problema i povećanju efikasnosti. Paralelno s povećanjem primjene povećava se i opseg istraživanja rada i svojstava genetskih algoritama i pokušavaju se svesti njihovi elementi na neke teorijske osnove. Nažalost, rezultati postignuti na teorijskom području su dvojbeni, a genetski algoritmi ostaju i do danas u osnovi heurističke metode.

Snaga genetskih algoritama, leži u činjenici da su oni sposobni odrediti položaj globalnog optimuma u prostoru s više lokalnih ekstrema, u tzv. višemodalnom prostoru. Klasične determinističke metode će se uvijek kretati prema lokalnom minimumu ili maksimumu, pri čemu on može biti i globalni, ali to se ne može odrediti iz rezultata. Stohastičke metode, tako i genetski algoritmi, nisu ovisne o nekoj eventualnoj početnoj točki i mogu svojim postupkom pretraživanja s nekom vjerojatnošću locirati globalni optimum određene ciljne funkcije. Osnovna razlika u primjeni između klasičnih i stohastičkih metoda je ta što za rezultat neke, recimo, gradijentne metode možemo sa sigurnošću reci da je postignut lokalni ekstrem unutar željene preciznosti. Za rezultat rada genetskog algoritma, međutim, nismo u mogućnosti sa stopostotnom vjerojatnošću reći da li predstavlja globalni ili samo lokalni optimum, te da li je isti određen sa željenom preciznošću. Koliko god se performanse stohastičkih metoda poboljšavale, one nikada neće moći dati niti jedan rezultat sa apsolutnom sigurnošću. Sigurnost dobivenih rezultata značajno se povećava postupkom ponavljanja procesa rješavanja, što kod klasičnih metoda nema smisla. Od kada su genetski algoritmi nastali, velika se pažnja poklanja istraživanjima vezanim za povećanje djelotvornosti izvedbe. Tijekom nešto više od dva desetljeća, a posebno u posljednjih nekoliko godina, pokazali su se vrlo moćnim.

4.1 Prikaz rješenja

Darwin je prije više od stoljeća ustvrdio da se vrste prilagođavaju okolišu kroz evoluciju. Stoljeće nakon Darwina, otkrivena je struktura kromosoma u kojima su zapisana sva svojstva jedinke. Ako se genetski materijal usporedi s abecedom, tada su slova u genetskom materijalu dušične baze adenin(A), timin(T), citozin(C) i gvanin(G). Sva živa bića imaju na jednak način zapisan genetski kod ovim nizovima dušičnih baza. Od ovih su dušičnih baza građene aminokiseline, njih oko 20 koje preuzimaju ulogu riječi u zapisu genskog materijala. Aminokiseline se redaju u lance i njihov slijed određuje sva

urođena svojstva jedinke. Ako su dušične baze slova evolucije, tada se aminokiseline mogu predstaviti kao riječi, a njihov niz kao potpuna knjiga o jedinki. U genetskom algoritmu rješenje postaje jedinka, a zapis rješenja mora se također prikazati u obliku sličnom prirodnom. Svi podaci koji obilježavaju jednu jedinku zapisani su u jednom kromosomu.

Na primjer, neka je problem jedno dimenzijski, odnosno traži se globalni optimum funkcije cilja jedne varijable *x* tako da vrijedi: *x* Î*[dg,gg]* gdje su *dg,gg* $\in \Re$. Tada jedna jedinka, odnosno jedan kromosom predstavlja jedno rješenje *x* $\in$ *[dg,gg]*. Podaci (u ovom slučaju je to jedan realan broj) mogu biti pohranjeni na razne načine u jedan kromosom. Na već uobičajen način i radi lakše prezentacije algoritma, neka su podaci spremljeni u niz bitova koji čine jedan kromosom. Ovakav prikaz (reprezentacija) naziva se binarni prikaz i opisan je u ovom poglavlju. Nadalje, jedan kromosom može biti i realan broj (u programskom jeziku *C* to je broj s pomičnom točkom jednostruke ili dvostruke preciznosti). Ako je problem višedimenzijski, umjesto jednog broja koji je zapisan u kromosomu, treba biti polje brojeva veličine dimenzije problema. Problemi mogu biti i druge prirode: npr. problem najkraćeg puta te problem rasporeda. U tom slučaju potencijalno rješenje zapisano u kromosomu je put ili raspored. U prvom slučaju kromosom može biti polje cijelih brojeva u kojem svaki broj označava redni broj mjesta, a niz brojeva označavaju put od početne točke do cilja.

Tablica 1. Primjer kromosoma označenog poljem cijelih brojeva

Kromosom 1	4826716141934
Kromosom 2	5479216468457

U drugom slučaju kromosom je dvodimenzijsko polje. Općenito, kromosom može biti bilo kakva struktura podataka koja opisuje svojstva jedne jedinke.

Tablica 2. Primjer označavanja realnim vrijednostima

Kromosom 1	A B E D B C A E D D
Kromosom 2	N W W N E S S W N N

Za genetski algoritam je značajno da kromosom predstavlja moguće rješenje zadanog problema. Za svaku strukturu podataka valja definirati genetske operatore. Međutim, genetski operatori trebaju biti tako definirani da oni ne stvaraju nove jedinke koje predstavljaju nemoguća rješenja, jer se time znatno umanjuje učinkovitost genetskog algoritma.

4.1.1 Prikaz rješenja s pomoću prirodnog binarnog koda

Prikaz rješenja može bitno utjecati na učinkovitost genetskog algoritma, stoga je izbor prikaza izuzetno značajan. Većina teorije vezane uz genetske algoritme je vezana upravo uz binarni prikaz. U praksi se pokazalo da binarni prikaz daje najbolje rezultate u većini primjera gdje se on može iskoristiti.
Kromosom kao binarni vektor predstavlja kodiranu vrijednost $x \in [dg, gg]$. Dužina *n* binarnog broja utječe na preciznost i označava broj bitova, odnosno broj jedinica ili nula u jednom kromosomu. U takav vektor je moguće zapisati 2^n različitih kombinacija nula i jedinica, tj. moguće je zapisati bilo koji broj u intervalu [0, 2^n-1]. Svaki bit u nizu predstavlja jednu karakteristiku rješenja.

Tablica 3. Primjer binarnog označavanja

Kromosom 1	1101100100110110
Kromosom 2	1101011000011110

4.1.2 Prikaz Grayevim kodom

Binarno kodiranje je zbog svoje jednostavnosti pogodno za implementaciju. Međutim, binarno kodiranje ima i jedan veliki nedostatak: Hammingova udaljenost među susjednim brojevima može biti velika, u najgorem slučaju jednaka duljini binarnog zapisa. Hammingova udaljenost između dva binarna broja je broj bitova u kojima se ta dva broja razlikuju. Na primjer, Hammingova udaljenost između brojeva $255_{10}=011111111_2$ i $256_{10}=100000000_2$ iznosi 9(svih devet bitova je različito). Drugim riječima, ako je genetski algoritam u nekom koraku pronašao jedno *dobro* rješenje za $b=011111111_2$, a optimum se postiže za $b=100000000_2$ treba promijeniti svih devet bitova. Kako bi se ispravio taj nedostatak, za kodiranje brojeva koristi se i Grayev kod.

Slika 6. Procedure za konverziju brojeva iz binarnog koda u Gray-ev kod i obrnuto

Susjedni brojevi kodirani u Grayevom kodu razlikuju se samo u jednom bitu. Dakle, Hammingova udaljenost između susjednih kodnih rijeci je 1.

4.2 Postupci selekcije

Svrha selekcije je čuvanje i prenošenje dobrih svojstava na slijedeću generaciju jedinki. Selekcijom se odabiru dobre jedinke koje će sudjelovati u slijedećem koraku, u reprodukciji. Na taj način se dobri geni ili dobri genetski materijal sačuvaju i prenose na slijedeću populaciju, a loši odumiru. Postupak selekcije bi se mogao ostvariti sortiranjem i odabirom *VEL_POP-M* najboljih jedinki (*VEL_POP* je veličina populacije, a *M* je broj jedinki određenih za eliminaciju). Međutim, takav postupak dovodi do prerane konvergencije genetskog algoritma, tj. proces optimiranja se praktično završava u svega nekoliko prvih iteracija. Problem je u tome što se ovim postupkom izgubi dobar genetski materijal koji mogu sadržavati loše jedinke. Zato je potrebno osigurati i lošim jedinkama da imaju neku (manju) vjerojatnost preživljavanja. S druge strane, bolje jedinke trebaju imati veću vjerojatnost opstanka, tj. trebaju imati veću vjerojatnost sudjelovanja u procesu reprodukcije. Genetske algoritme, s obzirom na vrstu selekcije, dijelimo na generacijske i eliminacijske. Generacijski genetski algoritam u jednoj iteraciji raspolaže s dvije populacije (što je ujedno i nedostatak generacijskog GA), jer se odabiru dobre jedinke iz stare populacije koje čine novu populaciju i nakon selekcije sudjeluju u procesu reprodukcije. Karakteristične vrste selekcija koje koristi generacijski GA su: jednostavna selekcija i turnirska selekcija. S druge strane eliminacijska selekcija je karakteristika eliminacijskog genetskog algoritma.

4.2.1 Jednostavna proporcionalna selekcija

Holland je 1975. godine predstavio jednostavni genetski algoritam s jednostavnom proporcionalnom selekcijom, jednostavnom mutacijom i križanjem s jednom točkom prekida. Vjerojatnost preživljavanja jedinke pri proporcionalnoj selekciji proporcionalna je njezinoj dobroti (fitnesu). Određena mjera kvalitete koja se u literaturi obično naziva dobrota ili fitnes, dok se funkcija koja tu kvalitetu određuje naziva funkcija cilja ili funkcija dobrote. Često se pri opisu proporcionalnih selekcija koristi analogija s kotačem ruleta. Analogija nije sasvim potpuna, jer svi brojevi na obodu kotača ruleta zauzimaju jednake kružne isječke, dok je zamišljena veličina kružnog isječka jedinke proporcionalna njezinoj dobroti, tj. vjerojatnosti njezine selekcije. Generacijska proporcionalna selekcija u svakom koraku generira slučajni broj r u intervalu [0,1]. Ako je r u intervalu [$F(i\text{-}1)$, $F(i)$], gdje je $F(x)$ već opisana funkcija razdiobe, tada je selektirana i-ta jedinka. Taj se postupak ponavlja sve dok se ne odabere N jedinki, s tim da se jedna te ista jedinka može odabrati proizvoljan broj puta. Vjerojatnost selekcije jedinke, tj. veličina njenog kružnog isječka, određuje se na temelju dobrote jedinke i ukupne dobrote populacije. Vjerojatnost selekcije i-te jedinke koja će sudjelovati u reprodukciji jednaka je kvocijentu dobrote jedinke i ukupne dobrote populacije.

4.2.2 Selekcija sortiranjem

Kod linearno sortirajuće selekcije vjerojatnost selekcije je proporcionalna rangu, odnosno poziciji jedinke u poretku jedinki sortiranih po dobroti. S tim da najbolja jedinka ima indeks N, a najgora 1. Vjerojatnost eliminacije za eliminacijsku linearno sortirajuću selekciju izračunava se prema istom izrazu, samo što su jedinke sortirane od najgore do najbolje, tj. najbolja jedinka ima indeks 1, a najgora N. Suma vjerojatnosti selekcija svih jedinki mora biti jednaka 1. Vjerojatnost selekcije najbolje ili najlošije jedinke može biti unaprijed zadana. Primjerice, vjerojatnost eliminacije najbolje jedinke treba biti jednaka nuli, ako se elitizmom štiti najbolja jedinka od eliminacije. Elitizam je mehanizam zaštite najbolje jedinke od bilo kakve izmjene ili eliminacije tijekom evolucijskog procesa.

4.2.3 Turnirska selekcija

Genetski algoritam s ugrađenom generacijskom turnirskom selekcijom u svakom koraku generira novu populaciju iz stare tako da *VEL_POP* puta odabire s jednakom vjerojatnošću k jedinki iz stare populacije, uspoređuje ih i najbolju jedinku kopira u bazen za reprodukciju nad kojim će u slijedećem koraku djelovati genetski operatori. Eliminacijska turnirska selekcija također

nasumice odabire *k* jedinki, ali eliminira najlošiju i nadomješta je s djetetom dviju (slučajno odabranih) preživjelih jedinki.

4.2.4 Eliminacijska selekcija

Generacijski genetski algoritam u jednom koraku raspolaže s dvije populacije jedinki: jednu populaciju dobiva iz prethodnog koraka, a drugu generira jednostavna selekcija. Kad selekcija generira novu populaciju iz prethodne, prethodna populacija se briše. Tek nakon što genetski operatori izvrše svoje djelovanje, novo nastala populacija je spremna za idući korak. To nepotrebno udvostručavanje populacije izbjegava se eliminacijskom selekcijom, odnosno eliminacijskom reprodukcijom. Za razliku od jednostavne selekcije, eliminacijska selekcija ne bira dobre kromosome za slijedeću populaciju, već loše koje treba eliminirati i reprodukcijom ih zamijeniti novima. Dakle, loši kromosomi umiru, a njih nadomještaju djeca nastala reprodukcijom roditelja, tj. preživjelih kromosoma.

4.3 Genetski operatori

Osim selekcije, reprodukcija je druga važna karakteristika genetskog algoritma. U reprodukciji sudjeluju dobre jedinke koje su preživjele proces selekcije. Reprodukcija je razmnožavanje s pomoću genetskog operatora križanja. Tijekom procesa reprodukcije dolazi i do slučajnih promjena nekih gena ili mutacije te zamjene gena ili inverzije.

4.3.1 Križanje

Križanje je operator koje iz dviju jedinki stvara novu, poželjno sa svojstvima obaju ishodišnih jedinki. Dvije jedinke koje su sudjelovale u nastanku nove nazivamo roditeljima, a novu jedinku djetetom. Rješenje se u ovom dijelu teksta naziva jedinka ili kromosomom po uzoru na biološko križanje (crossover) kromosoma. Prije križanja moguće je provjeravati da li su roditelji jednaki. U tom slučaju i dijete će biti jednako roditeljima. Moguće rješenje za uklanjanje duplikata je u tom slučaju mutirati jednog od roditelja.
Križanje s jednom točkom prekida nasumično odabire točku unutar kromosoma i izmjenjuje dio nakon točke prekida sa drugim roditeljem. Nije svejedno da li se uvijek zamjenjuje dio nakon ili prije točke prekida. Primjerice, ako se uvijek zamjenjuje dio nakon točke presjeka, dijete će uvijek imati početni dio od prvog roditelja. Rješenje je nasumično birati roditelje (poredak nije važan) ili nasumično odlučivati o tome koji dio se zamjenjuje. **Križanje s dvije točke prekida** također je često u praksi. Općenito križanje može biti s proizvoljno mnogo točaka prekida, dok je taj broj prekida manji od broja bitova. Uniformno križanje za svaki dio (bit) kromosoma djeteta bira hoće li ga dijete naslijediti od prvog ili drugog roditelja. Najčešće je vjerojatnost

nasljeđivanja od svakog roditelja jednaka, 50%, ali mogući su i pristupi u kojima vjerojatnost nasljeđivanja ovisi o funkciji ocjene za roditelje. Poluniformno križanje podvrsta je uniformnog križanja gdje se točno pola bitova koji se razlikuju u roditeljima zamjenjuju. Primijetimo da je poluniformno križanje slično uniformnom s vjerojatnošću 50%, ali nije identično.

4.3.2 Mutacija

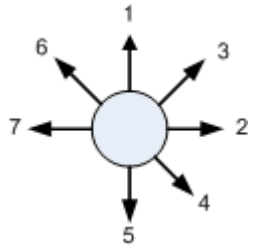
Zadaća je operatora mutacije unošenje raznolikosti u populaciju. Kao i u biologiji, mutacije se u najvećem broju slučajeva neće pokazati kao poboljšanje, ali su ukupno ključne za napredovanje populacije i izbjegavanje lokalnih maksimuma. Mutacijom se pretražuje prostor rješenja i upravo je mutacija mehanizam za izbjegavanje lokalnih minimuma. Naime, ako cijela populacija završi u nekom od lokalnih maksimuma, jedino slučajnim pretraživanjem prostora rješenja pronalazi se bolje rješenje. Dovoljno je da jedna jedinka (nastala mutacijom) bude bolja od ostalih, pa da se u nekoliko slijedećih generacija, sve jedinke presele u prostor gdje se nalazi bolje rješenje. Uloga mutacije je i također i u obnavljanju izgubljenog genetskog materijala. Dogodi li se, npr. da sve jedinke populacije imaju isti gen na određenom mjestu u kromosomu, samo križanjem se taj gen nikad ne bi mogao promijeniti. Ako je riječ o binarnom prikazu kromosoma, time je izgubljeno čak pola prostora pretraživanja. **Jednostavna mutacija** svaki dio (bit) kromosoma mijenja s određenom vjerojatnošću koja je parametar algoritma. **Miješajuća mutacija** zamjenjuje dijelove rješenja. Ukoliko se radi o binarnom prikazu, tada broj jedinica i nula ostaje isti. Ova mutacija ne dovodi do kvarenja rješenja u problemu trgovačkog putnika, gdje je važno da se element niza ne promijeni u neki koji u nizu već postoji, kako bi rješenje ostalo. **Invertirajuća miješajuća mutacija** dodatno invertira dijelove rješenja. Parametar mutacije p_M ne mora biti konstantan tokom izvođenja.

5. Programska implementacija

5.1 Prikaz rješenja

Genetski algoritam pisan je matematičkom programskom paketu MATLAB verzija 6.5. U daljem tekstu navodi se objašnjenje rada algoritma i opis problema koji su susretani prilikom izrade algoritma.

S obzirom na početnu i ciljnu poziciju potrebno pronaći prikladan broj točaka u pravokutnom kartezijevom sustavu koje bi označavale putanju robota. Robot se treba kretati putanjom u prostoru, a pritom ne uzrokujući sudare sa preprekama. Robot bi treba imati smjerove kretanja prema naprijed, nazad, lijevo, desno. Odgovarajuće kodiranje tih smjerova su izraženi brojevima 1,2,3,4,5,7 prikazanih na slici 5. Primijenjeno je 7 smjerova radi pojednostavljenja problema i lakšeg prikaza. Bude li potrebno lako se mogu implementirati dodatni smjerovi.



Slika 7. Smjerovi kretanja

Generira se početna populacija koja se sastoji od više rješenja. Potencijalno rješenje je prikazano kao **polje cijelih brojeva**, svaki broj označava korak u određenom smjeru, a niz brojeva označava put od početne točke do cilja. U MATLABU inicijalna populacija je nasumično definirana u obliku matrice koja je ograničena određenim brojem redaka i stupaca. Odnosno brojem kromosoma i veličinom kromosoma. Broj redaka odgovara broju kromosoma, broj stupaca veličini kromosoma.

Svaki redak matrice predstavlja jedno rješenje. Populacija ima slijedeću strukturu:

$$rand(E) = \begin{bmatrix} 1351.....4735 \\ ..... \\ 1264.....2424 \end{bmatrix} \quad (5.1)$$

Veličina kromosoma treba biti ograničena nekom funkcijom. To je prvi problem kojeg je trebalo riješiti tijekom izrade algoritma. Putanja se mora sastojati od dovoljnog broja točaka da bi se dosegla željena krajnja pozicija. Kako ograničiti broj koraka? Veličina kromosoma ovisi o ukupnoj udaljenosti koju robot treba prijeći i dužini jednog koraka (inkrement). Prema tome udaljenost između početne i ciljne pozicije je podijeljena sa dužinom jednog koraka prema izrazu (5.2).

$$A = \frac{\left|X_{start} - X_{cilj}\right| + \left|Y_{start} - Y_{cilj}\right|}{c} \quad (5.2)$$

A predstavlja dužinu kromosoma. Prilikom implementacije prepreka moralo se uzeti u obzir dodatni broj koraka potrebnih za izbjegavanje prepreka. Broj točaka je reguliran pomoću parametra ***c*** , koji se smanjivao ili povećavao s obzirom na dužinu između startne pozicije i cilja. Veličine parametra ***c*** dobiveni su iskustveno iz opsežnih testiranja i promatranja rezultata algoritma. Putanja se sastoji od skupa točaka. Točke su generirane tako da se na svaki korak dodaje inkrement po *x* i *y* osi u određenom smjeru. Naposljetku svaki korak tvori jednu točku u koordinatnom sustavu. Točke zajedno oblikuju vektor odnosno putanju kretanja.

5.2 Funkcija dobrote(fitnesa)

Funkcija ocjene odnosno fitnesa se temelji na najmanjoj udaljenosti postignutoj u trenutnoj generaciji. Najveća mogućnost preživljavanja se dodjeljuje rješenjima koja postignu najmanju udaljenost od cilja. Rješenja s većom dobrotom imaju veću šansu za prenošenje svojih shema u sljedeću generaciju. Prilikom ranije faze izrade algoritam uočena je neučinkovitost pronalaženja prikladne putanje. Taj problem je nastao zbog izjednačavanja dobrote kandidata prilikom izvođenja više iteracija(broja ponavljanja ili generacija). To dovodi do prerane konvergencije rješenja, tj. proces optimiranja se praktično završava u svega nekoliko prvih iteracija. Znači ako bi se u populaciji pojavilo bolje rješenje od prethodnog ono ne bi preživilo

zbog premale razlike u dobroti. Metodama skaliranja dobrote nastoji se interval vrijednosti dobrote, koji je definiran pojedincima populacije, po potrebi suziti ili proširiti kako bi se utjecalo na raznovrsnost izbora pojedinaca u reprodukcijsku populaciju. U zadatku dobrota ovisi o udaljenosti od cilja, problem skaliranja dobrote je riješen na način prikazan na slici 8.

```
novaudaljenost=udaljenost-
0.9*min(udaljenost);
suma=sum(novaudaljenost);
relUdaljenost = novaudaljenost./suma;
fitnes = 1-relUdaljenost;
fitnes=fitnes -0.95*min(fitnes);
```

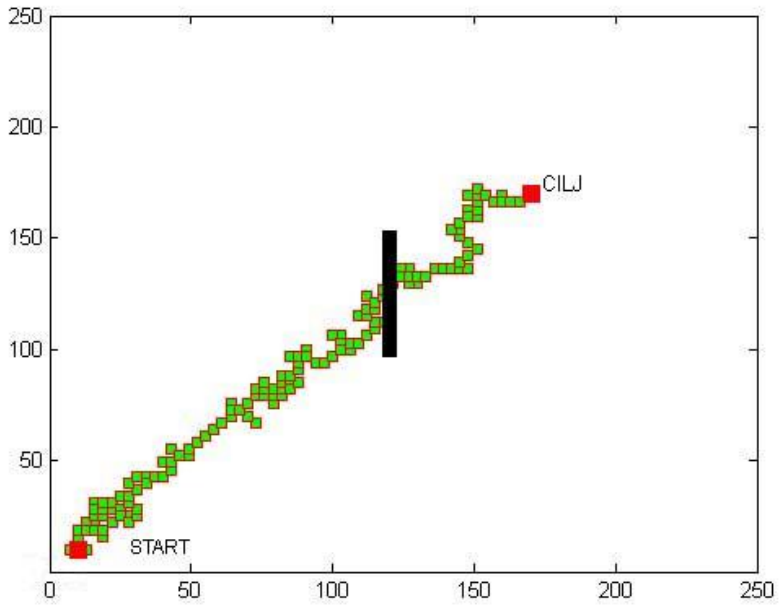
Slika 8. Izvorni kod operatora skaliranja fitnesa

5.3 Selekcija, mutacija i elitizam

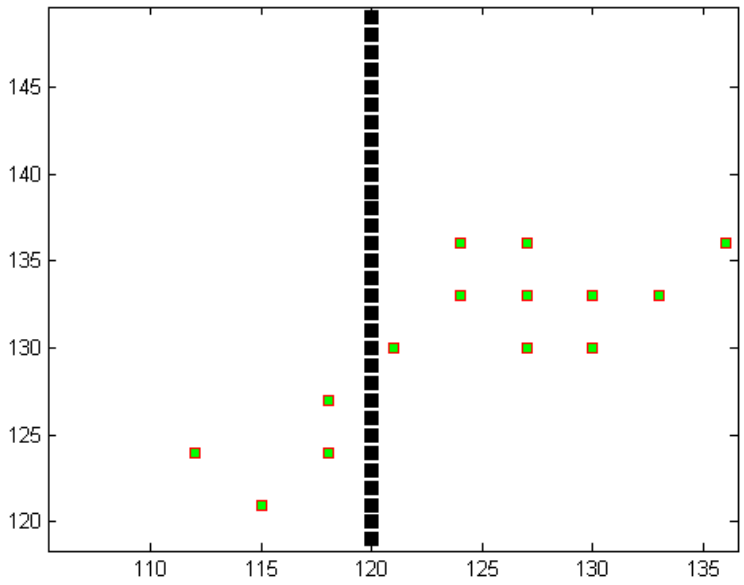
Koristi se jednostavna proporcionalna selekcija, jednostavna mutacija, križanje sa dvije točke prekida i vjerojatnošću križanja *pc*=0.7. Vjerojatnost mutacije *pm* je varijabillna. Ukoliko u određenom broju generacija rješenje ne konvergira prema cilju, tj. populacija zaglavi u lokalni optimum, vjerojatnost mutacije sa 3% povećava se na 40%. Takva vrijednost mutacije oslobađa populaciju iz lokalnog optimuma i dovodi do raspršenijeg pretraživanja.

Vjerojatnost preživljavanja jedinke pri proporcionalnoj selekciji proporcionalna je njezinoj dobroti. Novi problem je nastao ovim načinom dobivanja jedinki. Prilikom križanja i selekcije postoji mogućnost gubitka dobrog genetskog materijala. Iako najbolja putanja ima najveću vjerojatnost preživljavanja unatoč tome ona može izumrijeti. Prilikom testiranja algoritma primijećen je gubitak najboljih jedinki odnosno najboljih rješenja. Iz tog razloga potreban je mehanizam koji će zaštiti najbolju jedinku od izumiranja. Takav mehanizam se zove elitizam. U algoritmu elitizam je oblikovan tako da čuva najbolju jedinku, a najgoru jedinku zamjenjuje najboljom.

Prepreka se sastoji od skupa točaka u koordinatnom sustavu. Prilikom implementacije prepreke bilo je potrebno oblikovati funkciju koja bi osiguravala izbjegavanje prepreke. Funkcija primijenjena u algoritmu se temelji na kaznama. Kazne se dodjeljuju putanjama ako naiđu na prepreku. Takva kazna smanjuje dobrotu jedinke na 0.005 i samim time onemogućuje njenu selekciju te je izbacuje iz procesa reprodukcije. Unatoč kaznama, putanje su i dalje prolazile kroz prepreke kao što se vidi na slici 6. i 7. Bile su potrebne promjene na implementaciji prepreke i elitizmu.



Slika 9. Probijanje prepreke



Slika 10. Preskakanje prepreke

Putanja preskoči prepreku iz razloga jer korak(inkrement) između dvije točke iznosi 3, a prepreka je «debela» samo 1. Rješenje ovog problema je izvedeno tako da su proglašene dodatne dvije prepreke uz postojeću, tzv. sjene, koje se ne prikazuju na grafu. Time je onemogućeno preskakanje putanje preko prepreke. Funkcija elitizma teži za najboljom jedinkom tj. onom putanjom koja bi dospjela najbliže cilju u danoj iteraciji. Elitizam je trebalo prilagoditi kako bi se štitilo samo one putanje koje izbjegavaju prepreku, inače bi štitio i one putanje koje bi prolazile kroz prepreku.

5.4 Parametri algoritma

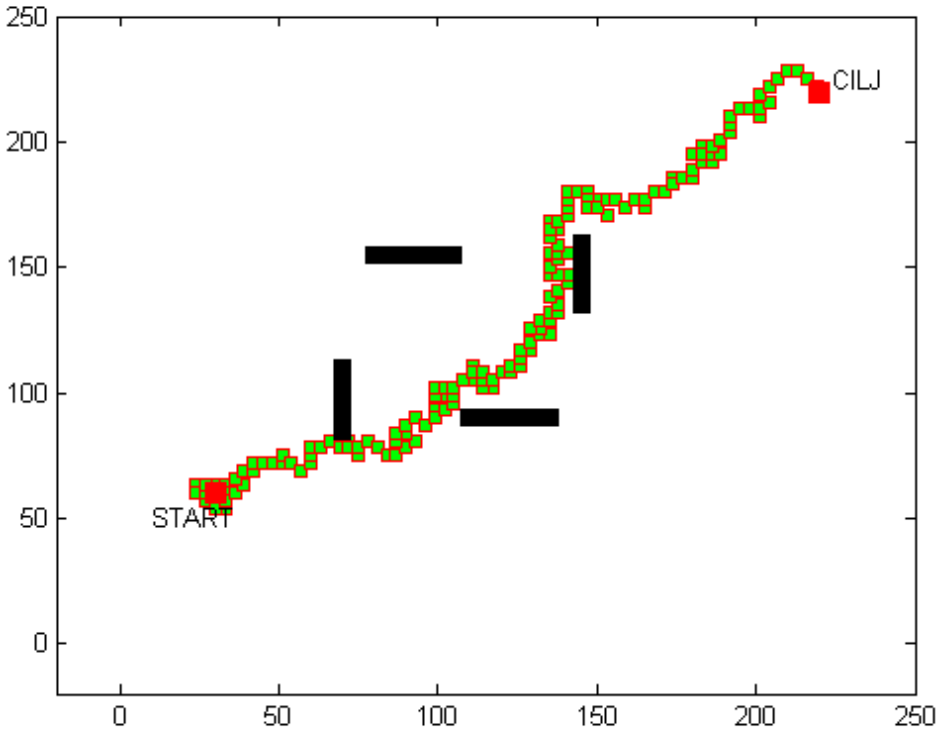
Algoritam se pri radu koristi konstantama čiju je vrijednost teško u potpunosti predvidjeti prije eksperimentalnog provjeravanja. Izvršeno je oko 600 testova. S obzirom na kvalitetu izlaza algoritma donosile su se vrijednosti parametara.

Tablica 4. Parametri algoritma

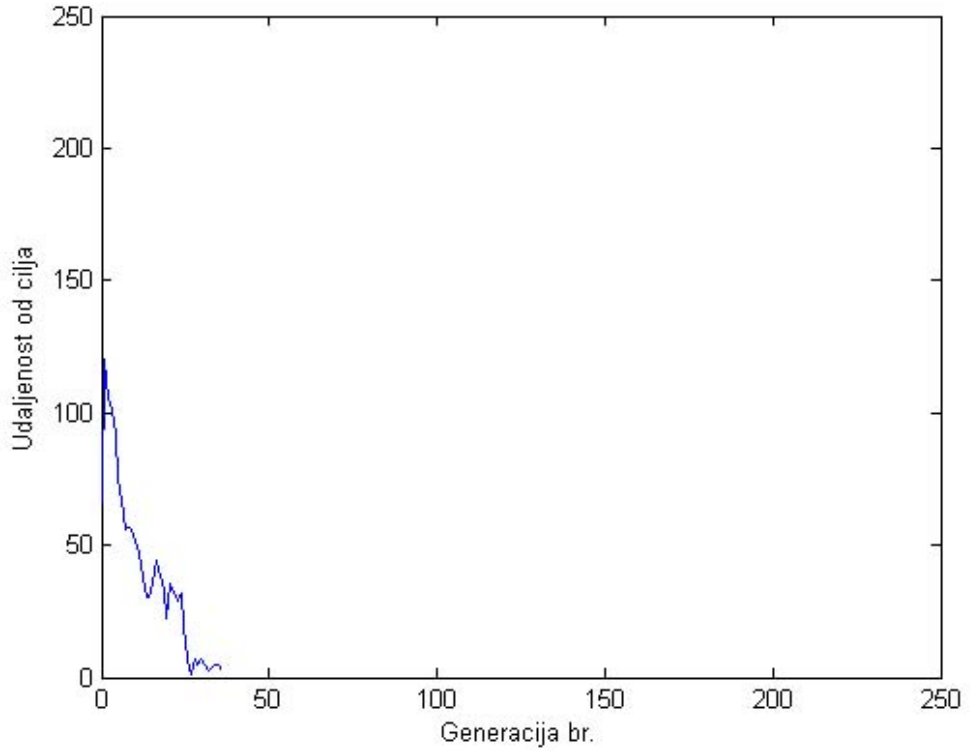
Naziv parametra	Objašnjenje	Vrijednost	Ispitane mogućnosti
B	Veličina populacije(broj kromosoma)	8 ,10	4, 5, 6,12, 15, 30
c	Faktor za računanje broja koraka	Ovisi o ukupnoj udaljenosti između startne i ciljne pozicije	0.9, 1,1.1, 1.6, 1.8 ,2.2 ,3
L	Broj iteracija	150	100, 200, 400, 500
pm	Određuje vjerojatnost mutacije pojedinog gena. Najbolja vrijednost za mutaciju nije ista za sve slučajeve	0.03, 0.4 za oslobađanje iz lokalnog optimuma	0.01,0.02,0.04
pc	Određuje vjerojatnost križanja dviju jedinki.	0.7	0.8, 0.6

5.5 Rezultati

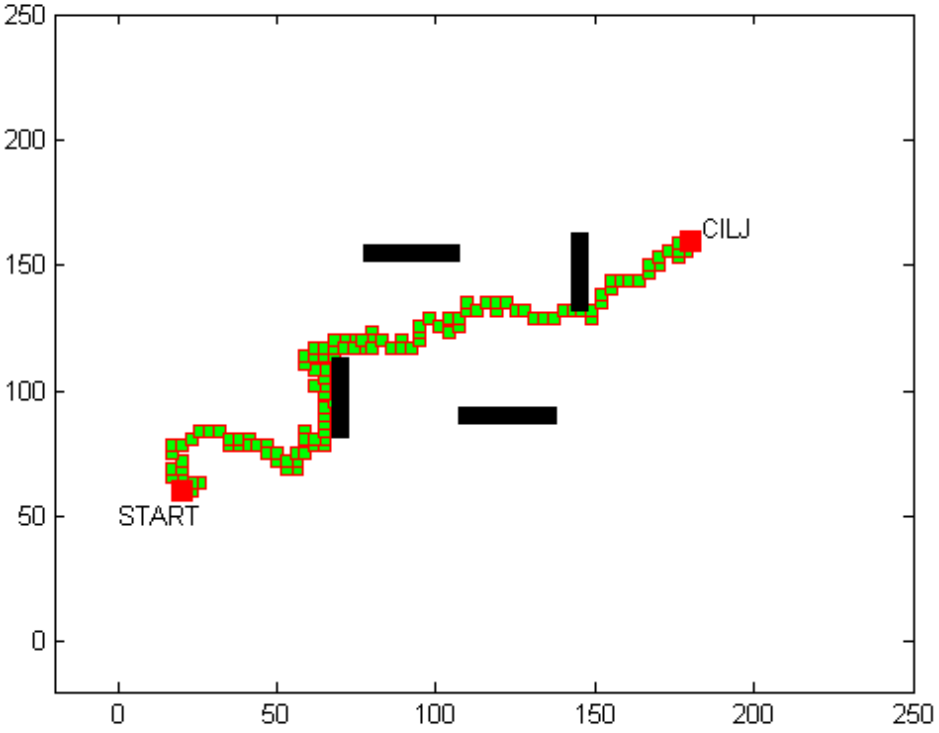
Za ovaj problem optimalna veličina populacije je 8. Uz tu populaciju vrijeme izvođenja algoritma je 1 pa do oko 4 minute za veće udaljenosti. Rješenja su dobivena u okviru 150 generacija. No u 90% slučajeva algoritam pronađe rješenje u 70 generacija. Treba naglasiti da se algoritam zaustavlja u trenutku dobivanja najboljeg rješenja.



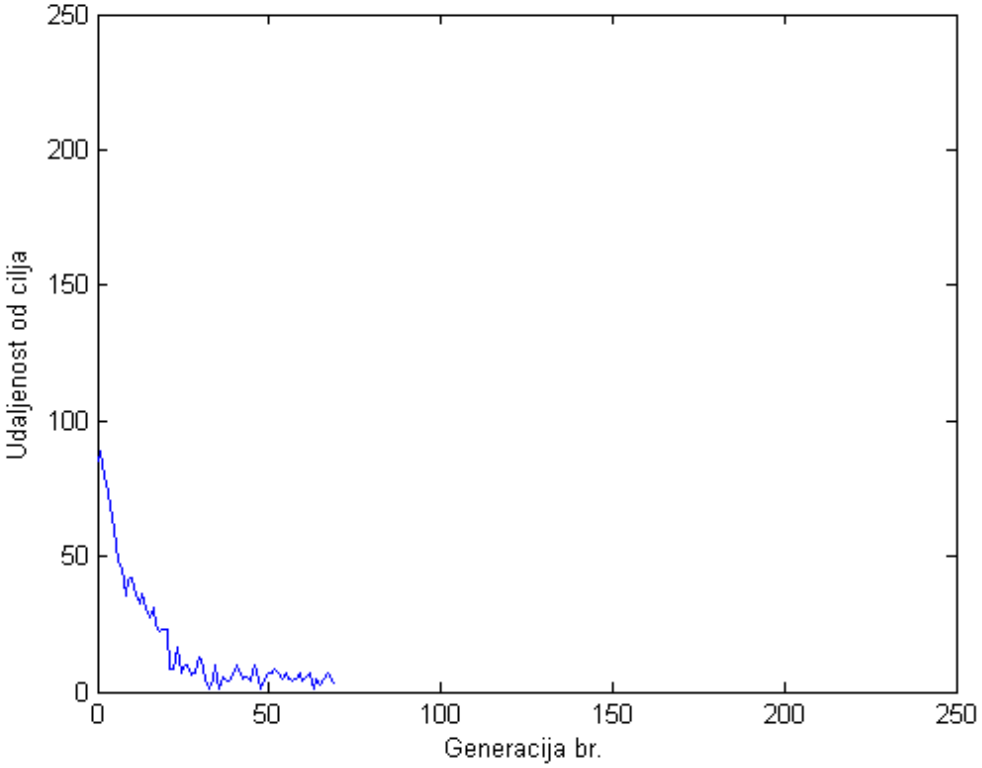
Slika 11. Slučaj 1



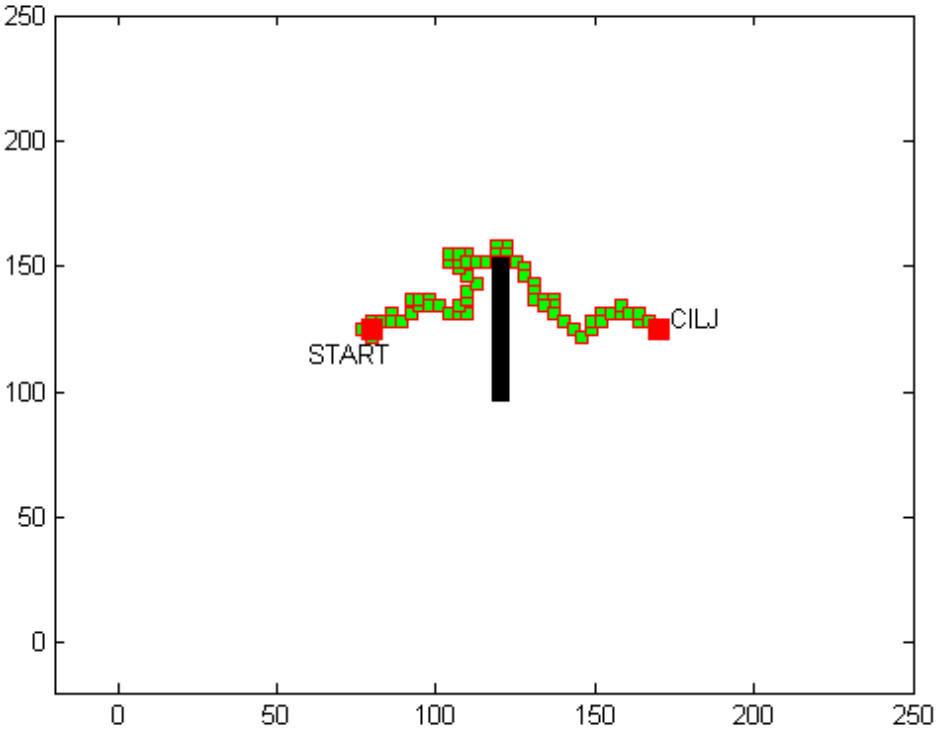
Slika 12. Konvergencija prema cilju za slučaj 1.



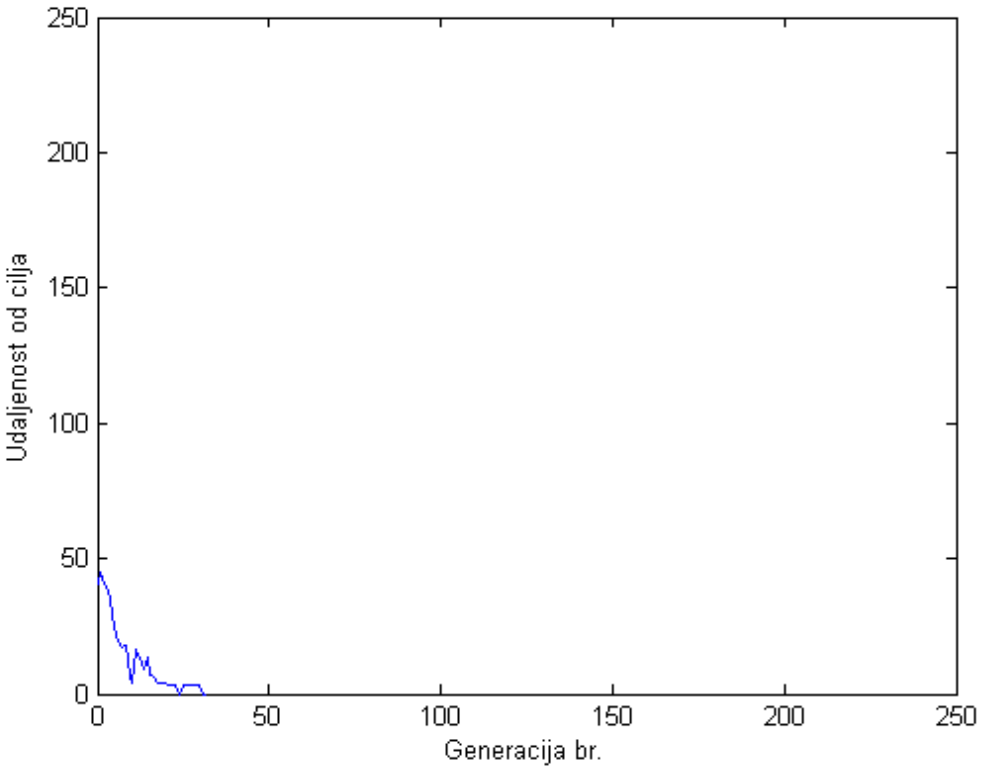
Slika 13. Slučaj 2



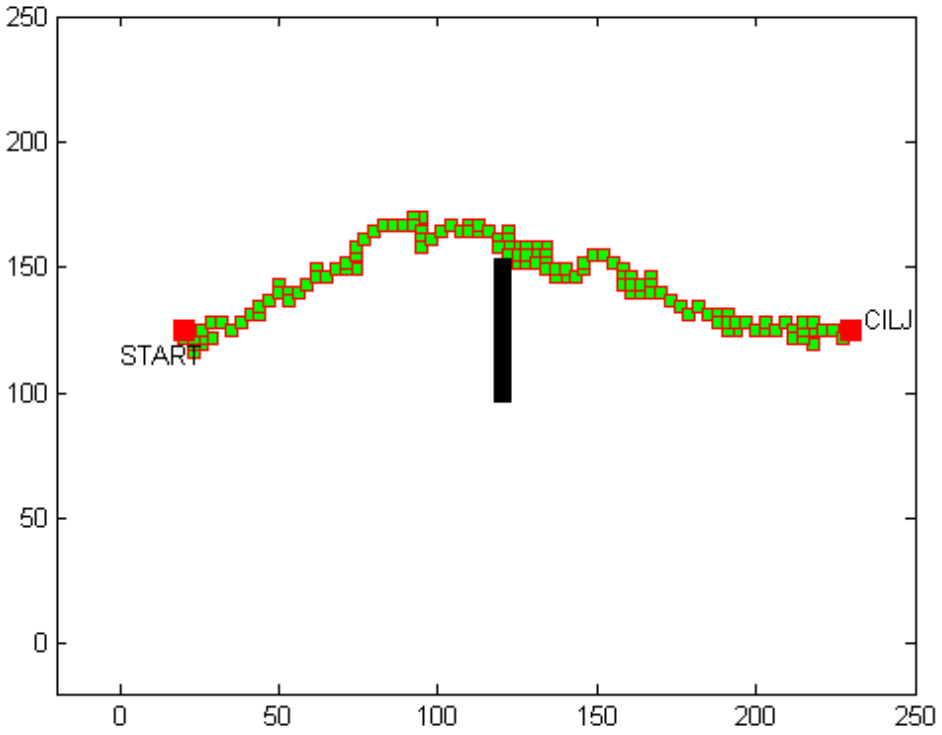
Slika 14. Konvergencija prema cilju za slučaj 2.



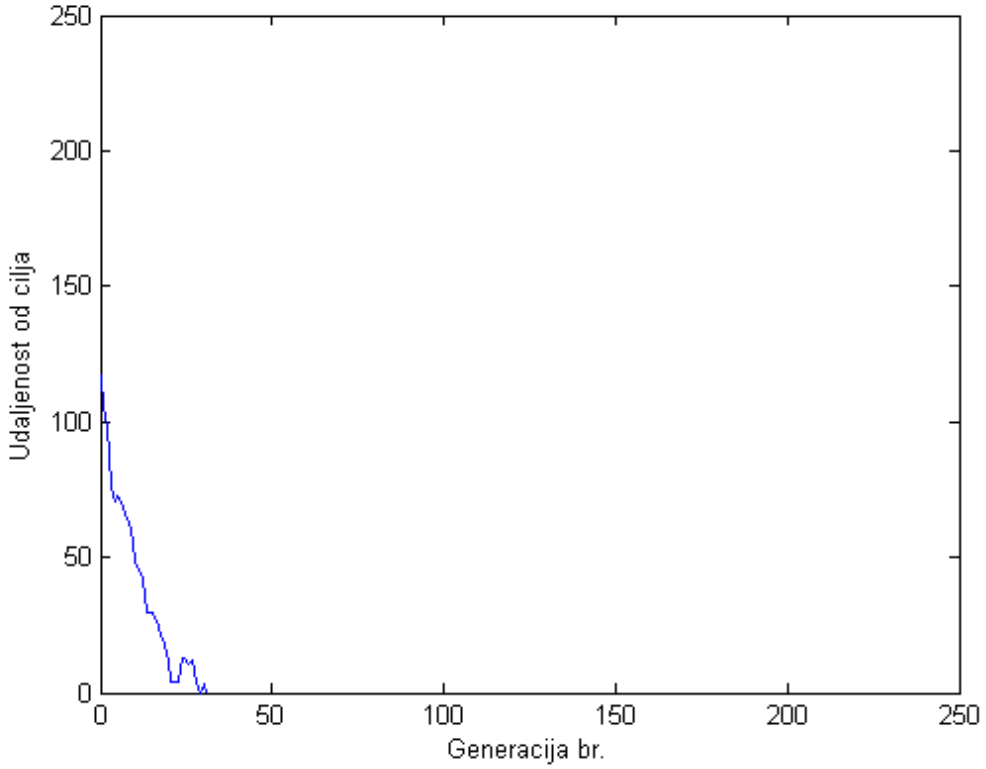
Slika 15. Slučaj 3.



Slika 16. Konvergencija prema cilju za slučaj 3.



Slika 17. Slučaj 4.



Slika 18. Konvergencija prema cilju za slučaj 4.

5.6 Kritički osvrt u odnosu na ostale metode i zaključak

PSO metoda se «udomaćila» u mnogim aplikacijama i pokazuje prednosti nad mnogim algoritmima. Njezina snaga je u tome što se uz male preinake osnovnog modela algoritma može napraviti algoritam koji učinkovito rješava različite vrste problema. Primjenu PSO metode možemo naći u različitim područjima poput neuronskih mreža i umjetne inteligencije. ACO metoda, zbog analogije na kojem je izgrađen, se uglavnom koristi u problemima koji se mogu svesti na pronalaženje najkraćeg puta u grafu no tijekom posljednjih godina primjenjuje se na raznim problemima. ACO metoda se u problemu pronalaska najkraćeg puta pokazala uspješnija i brža od GA. Metoda potencijalnog polja za brzo pronalaženje rješenja u izbjegavanju prepreka zbog problema sa lokalnim optimumom je napuštena. Odnosno primjenjuje se, ali sa implementiranim tehnikama optimizacije koje znatno poboljšavaju ovu metodu. Primjena umjetnih neuronskih mreža u robotici je široka, na području automatskog planiranja trajektorija robota, usavršavanja, napredak neuronskih mreža je nezaustavljiv.

U usporedbi s ostalim metodama optimizacije genetski algoritam ima dosta prednosti. Kao što je primjenjivost na više problema. Odlično aproksimira rješenja širokog spektra problema, pa su našli svoju primjenu u inženjerstvu, biologiji, ekonomiji, genetici, robotici, fizici, kemiji i mnogim drugim djelatnostima. Struktura algoritma nudi velike mogućnosti nadogradnje i povećanja efikasnosti algoritma jednostavnim zahvatima (puno stupnjeva slobode). Jednostavnim ponavljanjem postupka se može povećati pouzdanost rezultata, ako već ne nađe rješenje (globalni optimum), daje nekakvo dobro rješenje koje može zadovoljiti. Lošije strane genetskog algoritma bi bile: teškoća definiranja dobre funkcije cilja, sporija konvergencija naspram ostalih numeričkih metoda. Zbog izvođenja velikog broja računskih operacija GA je spor. Traži se velika procesorska snaga. Teško je postaviti dobre parametre(velik utjecaj parametara na efikasnost), potrebno je provesti puno eksperimentalnog provjeravanja. Zaključuje se kako su genetski algoritmi uz neke uvjete na problem moćan alat za rješavanje optimizacijskih problema pretraživanja.

6. Literatura

[1] Xiaohui Hu: ‘‘PSO tutorial”
Dostupno na adresi: http://www.swarmintelligence.org/tutorials.php
Datum nastanka: veljača 2006.

[2] G. Radanović: "Pregled heurističkih.algoritama"
Dostupno na adresi:http://www.zemris.fer.hr/~golub/ga/studenti/seminari/2007_radanovic/populacija.html
Fakultet elektrotehnike i računarstva, seminarski rad 2007, Zagreb

[3] Marco Dorigo, Mauro Birattari, i Thomas Stutzle:’’ Artificial Ants as a Computational Intelligence Technique’’
Dostupno na adresi: http://iridia.ulb.ac.be/~mdorigo/ACO/publications.html
Datum nastanka: 2006.

[4] H. Marković:" Primjena optimizacije kolonijom mrava na rješavanje problema trgovackog putnika’’
Fakultet elektrotehnike i računarstva, seminarski rad 2006, Zagreb
Dostupno na adresi:
http://www.zemris.fer.hr/~golub/ga/studenti/markovic/ACO.pdf

[5] Y. Koren, J. Borenstein: “Potential field methods and their inherent limitations for mobile robot navigation”
Dostupno na adresi: http://citeseer.ist.psu.edu/465483.html
Datum nastanka: travanj 1991.

[6] B.Novaković,D. Majetić,M. Široki: ’’Umjetne neuronske mreže”, Fakultet strojarstva i brodogradnje Sveučilišta u Zagrebu, 1998.

[7] D. Sutić: "Primjena genetskih algoritama"
Dostupno na adresi:
http://www.zemris.fer.hr/~golub/ga/studenti/seminari/2007_sutic
Fakultet elektrotehnike i računarstva, seminarski rad 2007, Zagreb

[8] Marin Golub: “Genetski algoritam: dio prvi”, verzija 2.3
Dostupno na adresi: http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf
Datum posljednje izmjene: 27. rujna 2004.

DODATAK A: Izvorni programski kôd

```
% Inicijalizacija programa

clear %Ponisti vrijednosti svih varijabli
clc
SUM=0;
y=0;
x=0;
inc= 3 ;         % [mm] udaljenost koja se dodaje pri jednom koraku po x,y koordinati

disp ('Definirajte inicijalnu populaciju')

Xc=input ('Unesite X koordinatu cilja: ');
Yc=input ('Unesite Y koordinatu cilja: ');

Xs=input ('Unesite X koordinatu pocetnog položaja: ');
Ys=input ('Unesite Y koordinatu pocetnog polozaja: ');

CILJ=[Xc Yc]   % zeljena koordinata cilja
START=[Xs Ys]    % startna pozicija

B=input ('Unesite broj kromosoma: (treba biti paran broj) ');
D=sqrt(((Xs-Xc)^2)+((Ys-Yc)^2))/1 %udaljenost izmedu startne pozicije i cilja

% Regulacija broja koraka sa parametrom c
if D<=100
   c=1.0;
elseif D>100
   c=1.1;
end
if D>=140
   c=1.25;
end
if D>=198
   c=1.3;
end
if D>=240
   c=1.33;
end
c;

A=round(sqrt(((Xs-Xc)^2)+((Ys-Yc)^2))/c) %velicina kromosoma ;  /

best=randint(1,A,[1,7]);
best_udaljenost=1000000;

L=input ('Broj iteracija: ');

%Vrijednosti vjerojatnosti križanja i mutacija

pc=0.7; , pm=0.03;

if mod(B,2)~=0
   disp( 'Greska! Broj gena mora biti paran broj!')
```

```
end

%Definicija inicijalne populacije
Populacija=randint(B,A,[1,7]);


% U varijable T i K upisuje broj redaka i stupaca inicijalizirane matrice
[T,K]=size(Populacija);
MM=T*K;
if mod(B,2)==0


   %Glavna petlja, ponavlja se koliko je definirano iteracija

   for MM=1:L

      for i=1:B

         for j=1:A

            trenutni=Populacija(i,j);

            if trenutni==1;
               X=inc;
               Y=0;

            elseif trenutni==2;
               X=0;
               Y=inc;
            elseif trenutni==3;

               Y=inc;
               X=inc;
            elseif trenutni==4;
               X=inc;
               Y=-inc;
            elseif trenutni==5;
               X=0;
               Y=-inc;
            elseif trenutni==6;
               X=-inc;
               Y=+inc;
            elseif trenutni==7;
               X=-inc;
               Y=0;
            end
            Ykor(i,j) = [Y];
            Xkor(i,j) = [X];

         end

         y=sum(Ykor')';% sumirane y koordinate po pojedinim recima (putanjama) ->doseg
         x=sum(Xkor')';% sumirane x koordinate po pojedinim recima (putanjama) ->doseg


         xy=[x,y];

      end
      % Dodana startna pozicija
```

```
y=y+START(1,2);
x=x+START(1,1);


%KOORDINATE PREPREKA


preprekay=[135:1:160];
prepreka_y=preprekay';
preprekax=[145:1:145];
prepreka_x=repmat(145,26,1);
x0y0=[prepreka_x,prepreka_y];


preprekay2=[85:1:110];
prepreka_y2=preprekay2';
preprekax2=[70:1:70];
prepreka_x2=repmat(70,26,1);
x2y2=[prepreka_x2,prepreka_y2];

preprekax1=[80:1:105];
prepreka_x1=preprekax1';
preprekay1=[155:1:155];
prepreka_y1=repmat(155,26,1);
x1y1=[prepreka_x1,prepreka_y1];

preprekax3=[110:1:135];
prepreka_x3=preprekax3';
preprekay3=[90:1:90];
prepreka_y3=repmat(90,26,1);
x3y3=[prepreka_x3,prepreka_y3];



for i=1:B

   udaljenost(i,1)=sqrt((Xc-x(i,1))^2+((Yc-y(i,1))^2)) % udaljenost izmedu cilja i dosega
putanje
end

% Fitnes i skaliranje fitnesa

novaudaljenost=udaljenost-0.9*min(udaljenost);
suma=sum(novaudaljenost);
relUdaljenost = novaudaljenost./suma;
fitnes = 1-relUdaljenost;
fitnes=fitnes -0.95*min(fitnes);


%Kaznjavanje putanja koje kolidiraju sa preprekom

for i=1:B

   pomak_x=START(1,1);
   pomak_y=START(1,2);



   for j=1:A
```

```
trenutni=Populacija(i,j);
if trenutni==1
   pomak_x=pomak_x+inc;

end
if trenutni==2
   pomak_y=pomak_y+inc;

end
if trenutni==3
   pomak_x=pomak_x+inc;
   pomak_y=pomak_y+inc;

end
if trenutni==4
   pomak_x=pomak_x+inc;
   pomak_y=pomak_y-inc;

end
if trenutni==5
   pomak_y=pomak_y-inc;

end
if trenutni==6
   pomak_x=pomak_x-inc;
   pomak_y=pomak_y+inc;

end
if trenutni==7
   pomak_x=pomak_x-inc;


end
kretanjex(i,j)=[pomak_x];
kretanjey(i,j)=[pomak_y];
trenutnix=kretanjex(i,j);
trenutniy=kretanjey(i,j);
korak=[trenutnix,trenutniy];


for k=1:26

   if korak==x0y0(k,:)
      fitnes(i)=0.005;

   elseif korak==(x0y0(k,:)+[2 0]) % OVDJE SAM DODAO [2 0] TAKO DA
PUTANJA NE BI "PRESKOCILA" PREPREKU
      fitnes(i)=0.005;
   elseif korak==(x0y0(k,:)+[1 0]) % OVDJE SAM DODAO [1 0] TAKO DA
PUTANJA NE BI "PRESKOCILA" PREPREKU
      fitnes(i)=0.005;

   elseif korak==x2y2(k,:)
      fitnes(i)=0.005;
   elseif korak==(x2y2(k,:)+[2 0])
      fitnes(i)=0.005;
   elseif korak==(x2y2(k,:)+[1 0]) % OVDJE SAM DODAO [1 0] TAKO DA
PUTANJA NE BI "PRESKOCILA" PREPREKU
      fitnes(i)=0.005;
```

```
            elseif korak==x1y1(k,:)
               fitnes(i)=0.005;
            elseif korak==(x1y1(k,:)+[0 2])
               fitnes(i)=0.005;
            elseif korak==(x1y1(k,:)+[0  1]) % OVDJE SAM DODAO [1  0] TAKO DA
PUTANJA NE BI "PRESKOCILA" PREPREKU
               fitnes(i)=0.005;

            elseif korak==x3y3(k,:)
               fitnes(i)=0.005;
            elseif korak==(x3y3(k,:)+[0 2])
               fitnes(i)=0.005;
            elseif korak==(x3y3(k,:)+[0  1]) % OVDJE SAM DODAO [1  0] TAKO DA
PUTANJA NE BI "PRESKOCILA" PREPREKU
               fitnes(i)=0.005;


            end
         end
      end
   end


   % Vjerojatnost izbora pojedinog kromosoma za razmnožavanje

   Vjerojatnost=fitnes/sum(fitnes);

   %  ELITIZAM - da li postoji put koji je bolji od najboljeg do sada pronadjenog (
best_udaljenost i best )
   % - cuvanje najbolje putanje

   for i=1:B
      if fitnes(i)>0.005
         if udaljenost(i,1)<best_udaljenost
            for j=1:A
               best(j)=Populacija(i,j);
            end
            best_udaljenost=udaljenost(i,1);
         end
      end

      max_udalj=max(udaljenost);
      for j=1:B
         if udaljenost(j)==max_udalj

            for k=1:A
               Populacija(j,k)=best(k);

            end
            udaljenost(j)=best_udaljenost;
            break;
         end
      end
   end
   % porast vjerojatnosti mutacije pm na 40% u slucaju lokalnog optimuma
   if MM>58
      if best_udaljenost>32
         pm=0.40;
      end
   end
```

```
if best_udaljenost<31
   pm=0.03;
end
% Generiranje parova za reprodukciju
for g=1:B
   r=rand(1,1); %Proizvoljna vrijednost za izbor kromosoma
   for k=1:B
      SUM=SUM+Vjerojatnost(k);
      if SUM>=r   %Uzimaju se dobre vrijednosti za roditelje (Rod=Roditelj)
         Rod(g,:)=Populacija(k,:);
         SUM=0;
         break
      end
   end
end
Rod;
%Ovdje se uzimaju parovi roditelja za reprodukciju
for h=2:2:B
   z=h-1;
   PAR_Trenutni=Rod(z:h,:);

   kk=rand(1,1); %Na temelju vjerojatnosti krizanja radi se krizanje

   if kk<=0.7

      qq=1+(A-1)*rand(1,1); %Odredjuje se breaking point za krizanje
      QQ=round(qq);

      jj=(A-QQ);
      AA=1+(jj)*rand(1,1);
      aa=round(AA);
      cc=QQ+aa-1;


      P1=PAR_Trenutni(1,:);
      P2=PAR_Trenutni(2,:);

      %Ovdje se dogadja to krizanje
      P11=[P1(1:QQ) P2(QQ+1:cc) P1(cc+1:A)];
      P22=[P2(1:QQ) P1(QQ+1:cc) P2(cc+1:A)];
      PAR_Trenutni_krizani=[P11;P22];

      %Ako nema krizanje, preslikaju se roditelji u novu populaciju
   else
      Potomak=PAR_Trenutni;
      PAR_Trenutni_krizani=Potomak;
   end

   % Novo=nova populacija
   Novo(z:z+1,:)=[PAR_Trenutni_krizani];

end
%Ovdje se dogadja mutacija
for tt=1:B


   for ttt=1:A
      pmut=rand(1,1);
      %Mutiram na trenutnom mjestu matrice u odnosu na vjerojatnost mutacije
```

```
            if pmut <= pm
                Novo(tt,ttt)=randint(1,1,[1 7]);

            end

        end


    end
    %Novo je ovdje vec krizana i mutirana populacija
    Populacija=Novo;

    najbolji(MM,1)=min(udaljenost);  % Za crtanje konvergencije

    % prekid rada u slucaju dobivanja najboljeg rješenja
    if best_udaljenost<=3
        break
    end


end


% VIZUALIZACIJA PUTANJE
najputanja=best;
pomak_x=START(1,1);
pomak_y=START(1,2);

for j=1:A

    trenutni=najputanja(j);
    if trenutni==1
        pomak_x=pomak_x+inc;

    end
    if trenutni==2
        pomak_y=pomak_y+inc;

    end
    if trenutni==3
        pomak_x=pomak_x+inc;
        pomak_y=pomak_y+inc;

    end
    if trenutni==4
        pomak_x=pomak_x+inc;
        pomak_y=pomak_y-inc;

    end
    if trenutni==5
        pomak_y=pomak_y-inc;

    end
    if trenutni==6
        pomak_x=pomak_x-inc;
        pomak_y=pomak_y+inc;

    end
    if trenutni==7
        pomak_x=pomak_x-inc;
```

```
        end

        pomakx(j)=[pomak_x];
        pomaky(j)=[pomak_y];


    end
end
set(figure,'Color','w')
plot(pomakx,pomaky,'sr','MarkerSize',5,'MarkerFaceColor','g')
hold on
plot(preprekax,preprekay,'sk','MarkerSize',8,'MarkerFaceColor','k')
plot(prepreka_x1,prepreka_y1,'sk','MarkerSize',8,'MarkerFaceColor','k')
plot(prepreka_x2,prepreka_y2,'sk','MarkerSize',8,'MarkerFaceColor','k')
plot(prepreka_x3,prepreka_y3,'sk','MarkerSize',8,'MarkerFaceColor','k')
plot(START(1),START(2),'sr','MarkerSize',10,'MarkerFaceColor','r')
plot(CILJ(1),CILJ(2),'sr','MarkerSize',10,'MarkerFaceColor','r')

axis([-20 250 -20 250])
text(CILJ(1,1)+4,CILJ(1,2)+4,'CILJ')
text(START(1,1)-20,START(1,2)-10,'START')

% VIZUALIZACIJA KONVERGENCIJE
set(figure,'Color','w')
XXX=linspace(0,MM,MM);
YYY=najbolji;
plot(XXX,YYY)
xlabel ('Generacija br.'),
ylabel ('Udaljenost od cilja');
axis([0 250 0 250])
end
```
